Sviluppo di workflow basati su reinforcement learning e digital twin per il supporto decisionale in ambito industriale

Università Ca' Foscari di Venezia
Dipartimento di Scienze Ambientali, Informatica e Statistica

Corso di laurea in informatica
Tesi di laurea
Anno 2023-2024

Laureando Enrico Brognera
Relatore Pietro Ferrara

Correlatore Francesco Mercuri

Ringraziamenti

Desidero innanzitutto ringraziare il relatore di questa tesi, il professor Pietro Ferrara, per la disponibilità e la gentilezza dimostrate durante tutto il periodo di tirocinio e la stesura della tesi. Un sentito ringraziamento va anche al professor Francesco Mercuri di Bologna, che mi ha sostenuto e mi ha dato la possibilità di svolgere un tirocinio molto interessante, permettendomi di conoscere una nuova realtà come il Centro Nazionale delle Ricerche (CNR).

Un grazie di cuore ai miei genitori, che finalmente posso ripagare almeno in parte dopo tutte le ansie e le preoccupazioni che ho dato loro e che meritavano di essere ricompensate. Un ringraziamento speciale ai miei fratelli: a Elena, per tutto il supporto e il sostegno emotivo, soprattutto nell'affrontare il mio ultimo esame, e a Matteo, per avermi supportato ogni giorno.

Infine, un sincero grazie a tutti i miei compagni di corso e ai miei amici, che, ognuno a modo suo, mi avete aiutato e lasciato qualcosa di prezioso. Sono tanti i ricordi che ci legano, ed è impossibile descriverli tutti a parole, ma so che mi accompagneranno e mi sosterranno sempre, così come lo hanno fatto in questi anni di università.

Abstract

In questa tesi si è sviluppato un workflow per un agente di Reinforcement Learning, con l'intento di fornire un supporto decisionale a un modello di Digital Twin, implementato in MatLab. Un Digital Twin è un modello virtuale di un sistema reale, utilizzato per simulazioni e ottimizzazioni. L'agente di Reinforcement Learning, attraverso l'analisi dei dati del Digital Twin e l'apprendimento dalle proprie azioni, fornisce un supporto decisionale che migliora l'efficienza del sistema.
Dopo aver descritto i concetti chiave del Machine Learning e del Reinforcement Learning, la tesi presenta lo sviluppo di un processo, o workflow, che comprende diverse attività (task). Queste attività includono la definizione dell'ambiente, con un'analisi di diversi scenari, focalizzandosi su un problema che introduce un modello di Digital Twin per l'ottimizzazione di un possibile sistema industriale. Si discute inoltre la definizione della reward, ovvero il riconoscimento che l'agente ottiene per le decisioni che migliorano le prestazioni del sistema, e la policy, che determina il comportamento dell'agente in base all'ambiente e alla reward.

Keywords Reinforcement Learning, Deep Learning, Workflow, Digital Twin, MatLab

Indice

Capitolo 1

Introduzione

Artificial intelligence (AI) refers to systems that display intelligent behaviour by analysing their environment and taking actions – with some degree of autonomy – to achieve specific goals. AI-based systems can be purely software-based, acting in the virtual world (e.g. voice assistants, image analysis software, search engines, speech and face recognition systems) or AI can be embedded in hardware devices (e.g. advanced robots, autonomous cars, drones or Internet of Things applications).[1]

Questa è la definizione fornita dalla commissione europea nel 2018 di Intelligenza Artificiale ma in realtà ha radici che risalgono agli anni '50, quando Alan Turing ha proposto il "Test di Turing" come misura dell'intelligenza di una macchina. Poi, nel 1956, John McCarthy ha coniato il termine "Intelligenza Artificiale" (in inglese *Artificial Intelligence*, da cui l'acronimo AI) alla Conferenza di Dartmouth. Da allora, l'AI ha continuato a svilupparsi e a espandersi, tuttavia, solo recentemente l'AI è diventata una componente fondamentale della nostra vita quotidiana. Questo è dovuto all'implementazione di AI in applicazioni come il filtraggio dello spam, la traduzione automatica, il riconoscimento e la creazione di contenuti multimediali, i sistemi di dialogo intelligenti. Un sottocampo dell'IA che ha guadagnato molta attenzione negli ultimi anni è il Machine Learning, o ML. Il ML utilizza algoritmi statistici per permettere ai computer di migliorare le loro prestazioni su un compito specifico con l'esperienza, cioè con l'apprendimento automatico da dati o informazioni senza essere esplicitamente programmati [10].

L'Intelligenza Artificiale e, in particolare, il Machine Learning, stanno rivoluzionando anche l'ambito dell'industria: permettono di automatizzare processi, migliorare l'efficienza e la produttività, e fornire supporto per la presa di decisioni strategiche. Possiamo quindi introdurre il termine di Industria 4.0, o quarta rivoluzione industriale, che rappresenta un'epoca di cambiamento radicale nella produzione, grazie all'integrazione di tecnologie digitali avanzate. Questo termine, nato in Germania, descrive la cooperazione di sistemi digitali e fisici, portando a un'automazione e controllo senza precedenti dei processi industriali. Un esempio significativo di questa rivoluzione è l'implementazione dei modelli Digital Twin. Un Digital Twin è una replica digitale di un sistema fisico, processo o prodotto che può essere utilizzato per simulare, prevedere e ottimizzare le performance [8]. Grazie all'AI e al Machine Learning, i Digital Twin possono essere integrati per costruire piattaforme di supporto decisionale, con l'obiettivo di migliorare la produzione. Utilizzando quindi la modellazione e la simulazione per prevedere e ottimizzare le prestazioni delle risorse e dei processi in vari scenari [2].

Questa tesi ha l'intento di esplorare e di sfruttare le potenzialità offerte dall'Intelligenza Artificiale e dal Machine Learning per migliorare e ottimizzare i processi industriali. In particolare, si concentra sull'apprendimento per rinforzo, una tecnica di Machine Learning che permette ai sistemi di apprendere e migliorare le proprie prestazioni attraverso l'interazione con l'ambiente e l'esperienza diretta. L'obiettivo principale di questa tesi è la creazione di un workflow che unisce il Machine Learning, in particolare l'apprendimento per rinforzo per generare un supporto decisionale per un modello Digital Twin in un ambiente industriale.

1.1 Introduzione al Machine Learning

Il Machine Learning (ML) è un sottoinsieme dell'intelligenza artificiale (AI) ed è definito come il campo di studio che dà ai computer la capacità di apprendere senza essere esplicitamente programmati [10], concentrandosi sull'utilizzo di dati e algoritmi per imitare il modo in cui gli esseri umani apprendono, migliorando gradualmente la loro accuratezza.
Il concetto di Machine Learning risale al 1943 con un'intuizione di Warren Mc-

Cullock e Walter Pitts, tuttavia i computer programmabili non esistevano ancora al momento della scoperta. A raccogliere la loro eredità e proseguire gli studi fu per primo lo psicoloco Frank Rosenblatt che costruì un dispositivo elettronico in grado di mostrare capacità di apprendimento chiamato *Perceptron*. Ma solo negli anni '80 iniziarono a essere sviluppate le prime reti neurali non lineari.

Esistono tre paradigmi principali nell'apprendimento automatico (ML) [9]:

- Apprendimento non supervisionato: è un tipo di ML in cui gli algoritmi scoprono autonomamente strutture interessanti nei dati senza risposte corrette predefinite. Questi algoritmi apprendono caratteristiche dai dati e, quando ne vengono introdotti di nuovi, utilizzano le caratteristiche apprese per riconoscere la diverse classe dei dati. È principalmente utilizzato per il clustering e la riduzione delle dimensioni. Il vantaggio principale è che può gestire grandi set di dati e scoprire relazioni nascoste tra le variabili senza la necessità di dati etichettati.

- Apprendimento supervisionato: è un tipo di ML che consiste nell'apprendere una funzione che mappa un input con un output basato su coppie di esempi input-output. Gli algoritmi di apprendimento supervisionato richiedono assistenza esterna. Il dataset di input viene diviso in un set di addestramento e uno di test, dove il set di addestramento contiene la variabile di output da prevedere o classificare. Questi algoritmi apprendono schemi dal set di addestramento e li applicano al set di test per predizioni o classificazioni. Il principale vantaggio dell'uso del ML supervisionato è che, una volta che un algoritmo impara cosa fare con i dati, può svolgere il suo lavoro automaticamente.

- Apprendimento per rinforzo (o Reinforcement Learning (RL)): è un'area di ML che imita il processo di apprendimento basato su tentativi ed errori utilizzato dagli esseri umani. Questi algoritmi vengono penalizzati quando prendono decisioni sbagliate e premiati quando prendono quelle giuste. Questo tipo di apprendimento è utilizzato per insegnare agli agenti software come prendere decisioni in giochi o in altri contesti che richiedono una serie di azio-

ni per raggiungere un obiettivo. Il vantaggio principale del Reinforcement Learning è la sua capacità di adattarsi a cambiamenti nell'input.

L'apprendimento per rinforzo è un paradigma completamente diverso. A differenza degli altri due framework di apprendimento che funzionano con un set di dati statico, RL funziona con un ambiente dinamico e l'obiettivo non è raggruppare i dati o etichettarli, ma trovare la migliore sequenza di azioni che genererà il risultato ottimale. Ottimale in questo caso significa raccogliere la ricompensa maggiore e lo fa consentendo a un pezzo di software chiamato agente di esplorare, interagire e imparare dall'ambiente. L'agente può intraprendere un'azione che influenza l'ambiente, modificandone lo stato, e l'ambiente produce quindi una ricompensa per quell'azione. Utilizzando queste informazioni, l'agente può modificare l'azione da intraprendere in futuro, imparando da questo processo.
Se prendiamo come esempio un robot che cammina, le osservazioni dell'ambiente potrebbero riguardare lo stato di ogni articolazione e le migliaia di pixel provenienti da un sensore fotografico. L'agente prenderebbe in considerazione tutti questi dati ed emetterebbe i comandi agli attuatori che permettono il movimento. Se il robot rimane in posizione eretta e continua a camminare, l'ambiente genererà una ricompensa, dicendo all'agente esattamente quanto bene ha funzionato quella combinazione molto specifica di comandi.

1.2 Stato dell'arte

Nell'era dell'Industria 4.0, l'importanza del Reinforcement Learning (RL) e dei Digital Twin è cresciuta esponenzialmente. Il RL, che ha visto un notevole aumento della letteratura negli ultimi anni [11], ha raggiunto trionfi significativi, superando i metodi tradizionali di controllo ottimale come RTO e MPC. Questo progresso è evidente in vari settori, dove l'RL è stato applicato con successo per migliorare il controllo dei processi, come dimostrato da AlphaGo Zero e AlphaStar di OpenAI [16].

Parallelamente, l'Industria 4.0 ha visto l'introduzione e l'importanza crescente dei Digital Twin. Questi modelli digitali, utilizzati per la simulazione di pezzi meccanici o macchinari in movimento in vari ambienti, hanno trovato applicazioni

significative. Un esempio notevole è la Boeing, che utilizza i Digital Twin nella progettazione dei suoi aerei. Questi modelli possono essere ulteriormente arricchiti utilizzando sistemi di machine learning e intelligenza artificiale per elaborare dati e generare nuova conoscenza [8].
L'integrazione di RL e Digital Twin ha portato a esempi di applicazioni innovative in vari settori [15]:

- Nel campo della manifattura, il digital twin è utilizzato per la produzione intelligente, la diagnosi dei guasti e l'ottimizzazione dei processi produttivi.
- Nel settore medico, i digital twin supportano la gestione delle prestazioni dei dispositivi medici e l'ottimizzazione del ciclo di vita degli ospedali.
- Nel settore dei trasporti, i digital twin sono impiegati per la gestione del traffico, la previsione della congestione e la manutenzione dei sistemi.

Contemporaneamente, il RL sta trovando applicazioni innovative in ambienti industriali per l'ottimizzazione di macchinari. Un esempio è uno studio del 2019 [14] in cui è stato proposto un possibile Digital Twin Framework per il controllo nella produzione petrolchimica che utilizza un approccio basato sul machine learning per formare un modello matematico di digital twin che simula gli input e output di controllo.
Un ulteriore esempio di come il RL può migliorare l'efficienza in un contesto industriale riguarda un macchinario proposto in un articolo del 2023 [5] per il packaging di prodotti alimentari. L'obiettivo era ottimizzare la velocità del nastro trasportatore, minimizzando la possibilità di lasciare scatole vuote nel nastro a causa della velocità di esso. Se si utilizzassero regole classiche per il packaging, sarebbe necessario inserire dei robot per gestire questa variabile. Tuttavia, utilizzando un algoritmo di RL, si è potuto migliorare l'efficienza del processo.

Capitolo 2

Fondamenti del Reinforcement Learning

Il RL è una metodologia di Machine Learning che cerca di replicare il processo naturale di apprendimento umano. Questo approccio si ispira ai primi anni di vita di un individuo, un periodo in cui l'esplorazione e la scoperta sono fondamentali per l'apprendimento. Durante questa fase, un bambino interagisce con il suo ambiente, sperimenta, commette errori e impara da essi. Questo processo di apprendimento attraverso l'interazione è al cuore dell'RL.
L'RL, quindi, è un approccio computazionale all'apprendimento che si basa sull'interazione con l'ambiente. Invece di essere programmato con regole e istruzioni specifiche, un algoritmo di RL impara attraverso l'esperienza. Esplora l'ambiente, prende decisioni, riceve feedback sotto forma di ricompense o punizioni e usa queste informazioni per migliorare le sue future decisioni. Questo processo di apprendimento continuo e adattativo mira a ottimizzare le prestazioni dell'algoritmo nel tempo [6].

2.1 Elementi di RL

La figura 2.1 illustra l'architettura tipica del RL, questa struttura è composta principalmente da due elementi: l'agente (o **Agent**) e l'ambiente (o **Environment**). A ogni passaggio temporale, l'agente si trova in uno stato specifico (o **State**) e può eseguire delle azioni (o **Action**). Queste azioni possono appartenere a insiemi discreti o continui e possono avere più dimensioni.

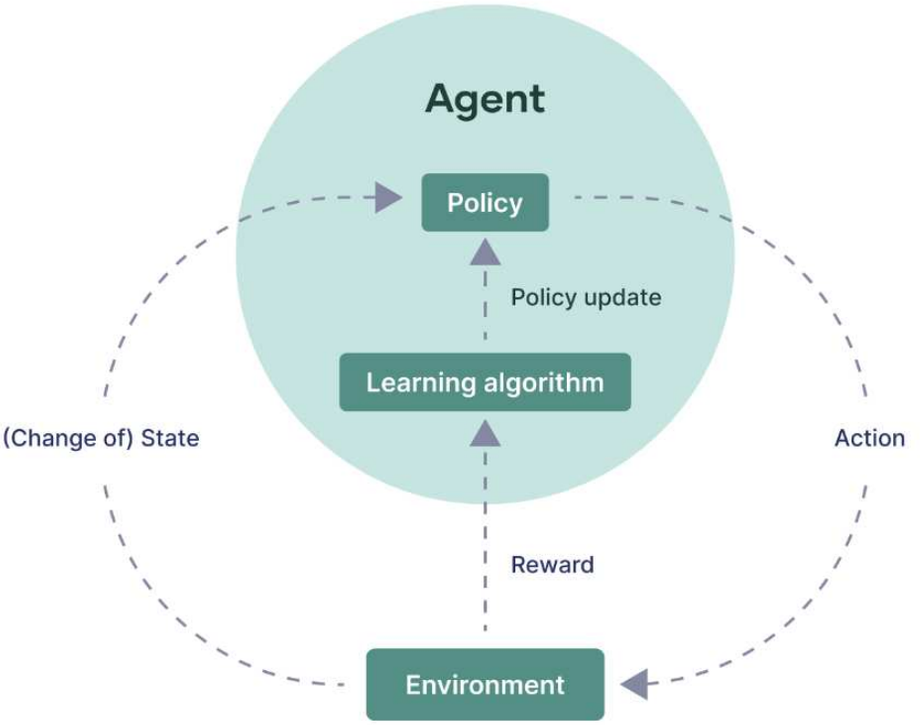

Figura 2.1: RL Workflow

Quando l'agente esegue un'azione, lo stato dell'ambiente cambia e questa variazione viene valutata dall'ambiente stesso. In risposta a ogni azione, l'ambiente fornisce all'agente una ricompensa (o **Reward**), rappresentata con un singolo valore scalare. L'obiettivo principale dell'agente sarà quello di massimizzare la somma delle ricompense ricevute nel lungo periodo.
Durante l'interazione con l'ambiente, l'agente mirerà a trovare una politica, che mappa gli stati in azioni, al fine di massimizzare il rendimento complessivo. Questa politica permette all'agente di prendere decisioni ottimali basate sullo stato corrente dell'ambiente.

2.1.1 Environment

Un **RL Environment** è il contesto in cui opera l'agente RL. Si tratta di un ambiente fisico o una simulazione con cui l'agente interagisce, eseguendo azioni, ottenendo ricompense e passando a nuovi stati in base a queste azioni. L'ambiente offre all'agente il suo stato iniziale e lo stato attuale dopo ogni azione. Inoltre, stabilisce la ricompensa associata a ogni coppia di stato-azione [18].

Gli RL Environment sono fondamentali per l'addestramento degli agenti RL, poiché

forniscono il ciclo di feedback necessario, permettendo all'agente di apprendere dalle sue azioni e di affinare la sua politica nel tempo. La complessità e la varietà di questi ambienti possono avere un impatto significativo sul processo di apprendimento e sulle prestazioni dell'agente addestrato.
In un RL Environment, l'agente inizia in uno stato iniziale. Successivamente, intraprende un'azione basata sulla sua politica corrente e, in risposta, l'ambiente fornisce all'agente un nuovo stato e una ricompensa. Questo processo di stato-azione-ricompensa (Fig 2.1) prosegue fino a quando non viene raggiunto uno stato finale.

Esempi di ambienti di apprendimento per rinforzo includono simulazioni di giochi (come scacchi, Go o videogiochi come StarCraft o giochi Atari), simulazioni di controllo robotico (come la palestra di OpenAI) o addirittura ambienti reali (come veicoli a guida autonoma o sistemi industriali).
L'uso di un ambiente simulato offre diversi vantaggi rispetto a uno fisico. Innanzitutto, l'apprendimento necessita di numerosi tentativi; in questo contesto, le simulazioni accelerano il processo, consentendo l'esecuzione parallela di diverse simulazioni. Inoltre, esse offrono la possibilità di riprodurre condizioni che sarebbero complesse da sperimentare nel mondo reale.

2.1.2 Reward

Il **reward** è il valore che definisce il vantaggio immediato derivante dall'essere in uno stato specifico, mentre il **value** è la somma totale dei reward futuri previsti. Uno stato può avere un reward immediato basso ma un value alto se è seguito da stati con reward alti.
Il reward può essere una funzione stocastica, una funzione lineare o una sequenza di diverse costanti [4].
La creazione di una funzione di ricompensa efficace è uno degli aspetti più sfidanti nel RL, ci sono due problemi principali:

- Ricompensa Sparsa (Sparse Reward): Quando l'obiettivo che si vuole incentivare arriva dopo una lunga sequenza di azioni, l'agente può inciampare per lunghi periodi di tempo senza ricevere alcuna ricompensa, questo può rende

molto improbabile che l'agente scopra casualmente la sequenza di azioni che produce la ricompensa.

- Modellazione della Ricompensa (Reward Shaping): Fornire ricompense intermedie più piccole può ridurre il problema citato precedentemente, tuttavia, la modellazione della ricompensa può comportare una serie di problemi, poiché se si fornisce una scorciatoia l'algoritmo la prenderà. Questo può portare l'agente a convergere in una soluzione sub ottimale, ma non ideale.

Un altro aspetto critico dell'apprendimento per rinforzo è il compromesso tra **exploration** e **exploitation** quando si interagisce con l'ambiente. Questo è la differenza tra raccogliere il maggior numero di ricompense che già conosci (exploitation) ed esplorare aree dell'ambiente che non hai ancora visitato (exploration).

2.1.3 Policy

La **policy** definisce il modo con cui l'agente apprende e il suo comportamento in un dato istante [4]. Per semplificare, possiamo considerare la policy come una mappatura tra gli stati che l'ambiente percepisce e le azioni da intraprendere quando ci si trova in tali stati. A volte, la policy può essere espressa come una funzione semplice o una tabella di lookup, mentre in altre situazioni può richiedere calcoli più complessi, come processi di ricerca. La policy costituisce il cuore di un agente di apprendimento per rinforzo (RL), poiché determina autonomamente il suo comportamento. In generale, le policy possono essere deterministiche, basandosi esclusivamente sullo stato, oppure stocastiche, definendo le azioni attraverso distribuzioni di probabilità, date le condizioni di stato.

2.2 Markov decision process

Il Markov decision process (MDP) [13] fornisce il quadro matematico per l'RL. Un MDP è formalmente definito dai seguenti componenti:

- S: Insieme finito di stati.

- A: Insieme finito di azioni.

- $P(P : S \times A \to \Pi(S))$: Funzione di transazione che assegna una distribuzione di probabilità a ogni coppia stato-azione.

- $R(R : S \times A \times S \to \mathbb{R})$: Funzione di rinforzo o Reward che assegna un valore numerico a ogni possibile transizione.

Per la proprietà di Markov che ne deriva all'istante t uno stato s_t è considerato **Markoviano** se e solo se la probabilità della transazione verso un successivo stato s_{t+1} non dipende dalla storia completa degli stati precedenti ma solo dallo stato corrente s_t. Quindi dato lo stato corrente, il futuro è indipendente dal passato. La proprietà di Markov riveste un ruolo fondamentale nel reinforcement learning perché i metodi correlati a questo tipo di apprendimento si basano sull'assunzione che i valori forniti dall'ambiente dipendano esclusivamente dallo stato corrente e dall'azione intrapresa nell'istante precedente.

2.3 Valore di Ritorno e Funzione Valore

Il **valore di ritorno** è la misura del beneficio ottenuto da un'azione in un MDP, quindi è la somma delle ricompense scontate nel tempo. Il valore di ritorno al tempo t è formalmente definito come:

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{2.1}$$

Dove R_t è la ricompensa (Reward) al tempo t e $\gamma \in [0, 1)$ è il **fattore di sconto** che determina l'importanza delle ricompense future rispetto a quelle immediate.
La **funzione valore di stato** $V(s)$ assegna un valore numerico a ciascuno stato in un MDP, che indica quanto questo è "buono" in termini di valore di ritorno atteso. La funzione $V(s$ è quindi la somma delle ricompense attese partendo da uno stato s.

$$V(s) = E\left(G_t | s_t = s\right) = E\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s\right) \tag{2.2}$$

Dove E rappresenta il valore atteso.
Similmente la **funzione azione-valore** $Q(s, a)$, chiamata anche **Q-function**, è

definita come la somma delle ricompense previste da un'azione a in uno stato s.

$$Q(s,a) = E\left(G_t | s_t = s, a_t = a\right) = E\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, a_t = a\right) \tag{2.3}$$

2.4 Equazione di Bellman

L'equazione di Bellman è utilizzata per formulare il problema di ottimizzazione nei Markov Decision Process ed è quindi alla base di molti algoritmi di RL. Il suo obiettivo è quello di massimizzare il valore atteso totale accumulato nel lungo termine partendo da uno stato s [18]. Formalmente:

$$V(s) = \max_a \left(R(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s') \right) \tag{2.4}$$

Dove a rappresenta le azioni possibili nello stato s, $R(s,a)$ è la ricompensa immediata ottenuta eseguendo l'azione a nello stato s e $P(s'|s,a)$ è la probabilità di transizione dallo stato s allo stato s' dopo aver eseguito l'azione a.
Questa stessa equazione può essere adattata anche alla funzione azione-valore (2.3)

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a') \tag{2.5}$$

Dove $\max_{a'} Q(s',a')$ rappresenta il valore massimo della funzione Q per tutte le possibili azioni a' nello stato successivo s'.

2.5 Q-Learning

Il Q-learning è uno dei più popolari algoritmi di RL. Si basa sull'utilizzare una funzione Q per ottenere il valore utilizzato per trovare la politica ottimale di selezione delle azioni; tali valori vengono salvati e aggiornati tramite la **Q-Table**. La Q-table calcola la ricompensa futura massima prevista per ciascuna azione in ciascuno stato.
La funzione Q utilizza l'equazione di Bellman (2.3) per calcolare i valori di tutte le celle della Q-Table. Inizialmente saranno tutti pari a zero e quando inizieremo a esplorare l'ambiente, questi valori verranno aggiornati fornendo approssimazioni sempre migliori [18]. Questi aggiornamenti seguono il seguente algoritmo:

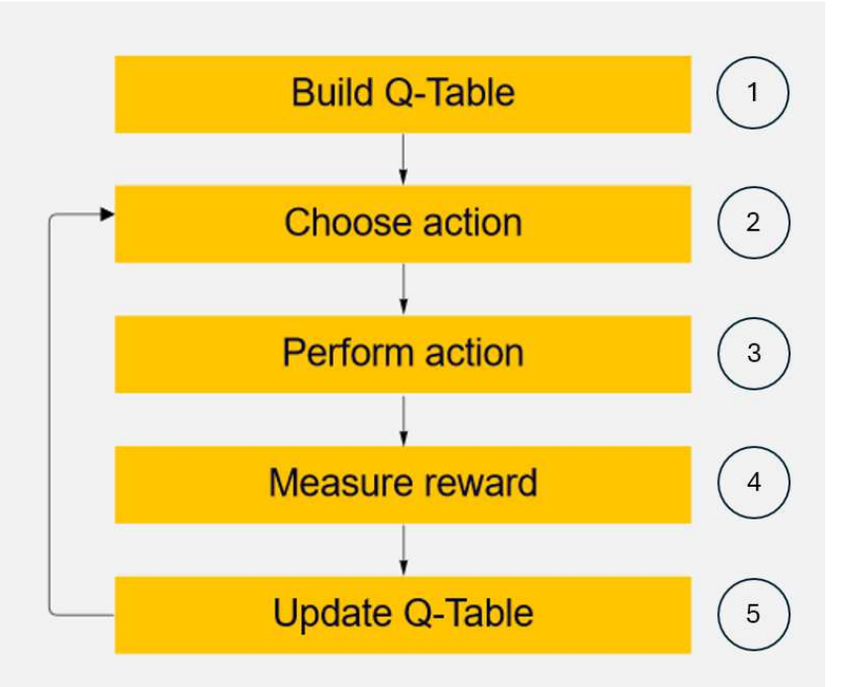

Figura 2.2: Algoritmo di aggiornamento della Q-Table

I passaggi da due a cinque verranno ripetuti più volte in un ciclo iterativo finché non viene creata una "buona" Q-Table.

Nel primo passaggio l'algoritmo inizializza tutti i valori della Q-Table, Q-Table formata da n colonne, dove n = numero di azioni, e m righe, dove m = numero di stati.

Nei passaggi 2 e 3 verrà scelta dall'agente un'azione a in base alla sua politica, per esempio in maniera casuale oppure in base al miglior valore Q. Un'altra strategia è la **Epsilon Greedy Strategy** che è un metodo semplice per bilanciare l'esplorazione (**exploration**) e lo sfruttamento (**exploitation**) delle azioni, quindi più l'epsilon è grande, più l'agente è orientato all'esplorazione.

Dopo che è stata intrapresa un'azione e osservato un risultato, l'ultima operazione rimanente è l'aggiornamento del valore nella Q-Table. Tale aggiornamento seguirà la seguente equazione:

$$Q_{new}(s,a) = Q(s,a) + \alpha \cdot [R(s,a) + \gamma \cdot \max Q'(s',a') - Q(s,a)] \tag{2.6}$$

Dove:

- $Q(s,a)$ è il valore Q corrente
- α è il *Learning Rate* o tasso di apprendimento

- $R(s, a)$ è il valore della Reward immediata

- γ è il *Discount Rate* o tasso di sconto che determina di quanto l'agente si preoccupa delle ricompense nel lontano futuro rispetto a quelle nell'immediato.

- $\gamma \cdot \max Q'(s', a')$ è la stima del valore Q ottimale dello stato successivo

2.6 Deep Reinforcement Learning

Gli algoritmi di Deep Reinforcement Learning (DRL) hanno suscitato un notevole interesse all'interno della comunità di AI (Artificial Intelligence) negli ultimi anni [11]. Con il termine "Deep Reinforcement Learning" ci si riferisce semplicemente all'utilizzo di reti neurali profonde (o **Deep Neural Networks**) come approssimatori di funzione per la funzione valore o la policy in algoritmi di Reinforcement Learning.
Una Deep Neural Networks (DNN) è un insieme di livelli, ciascuno costituito da un insieme di nodi che operano insieme per eseguire un'attività specifica. Solitamente queste reti sono composte da uno strato di input, uno o più strati nascosti e uno strato di output; ogni livello aggiunge complessità al modello, consentendo al contempo di elaborare gli input in modo più preciso per produrre la soluzione ideale.

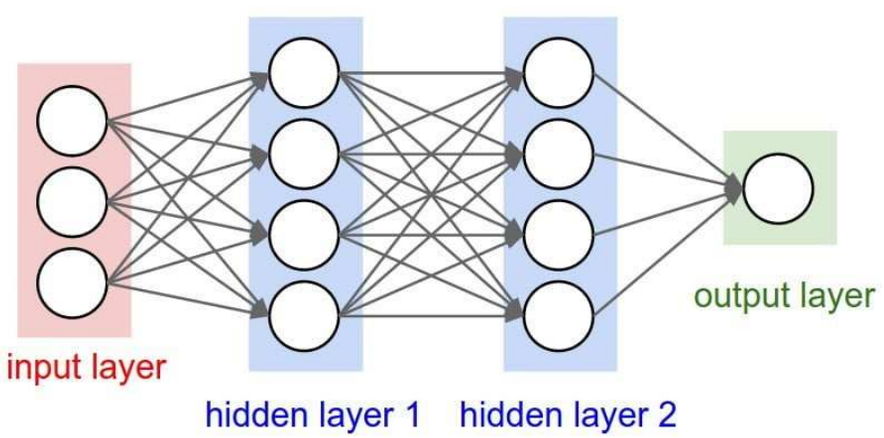

Figura 2.3: Esempio di una DNN con due livelli nascosti

2.6.1 Deep Q-Learning

Il Deep Q-Learning utilizza l'idea del Q-learning e fa un ulteriore passo avanti; invece di utilizzare una Q-Table, utilizza una Deep Neural Network che prende in input uno stato e approssima i valori Q per ciascuna azione basata su quello stato.

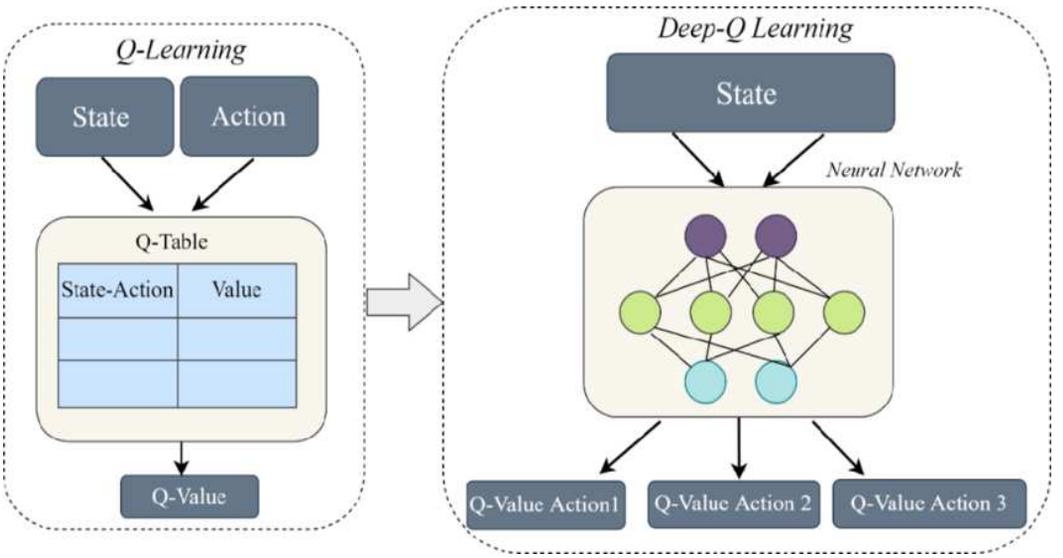

Figura 2.4: Confronto tra Q-Learning e Deep Q-Learning

L'utilizzo di una Q-Table non è molto scalabile perchè con modelli complessi con un elevato numero di possibili azioni e stati la Q-Table diventa velocemente complessa e difficile da risolvere in modo efficiente. Quindi una soluzione è l'uso di una DNN che riceve lo stato come input e produce Q-value diversi per ciascuna azione. Quindi la scelta dell'azione da parte dell'agente può essere la medesima del Q-learning e il processo di apprendimento rimane sempre lo stesso con l'approccio di aggiornamento iterativo, ma invece di aggiornare la Q-Table, aggiorneremo i pesi nella rete neurale in modo che gli output siano migliorati.

2.6.2 Proximal Policy Optimization

Il Proximal Policy Optimization (PPO) è attualmente considerato lo stato dell'arte del Reinforcement Learning. L'algoritmo, introdotto da OpenAI nel 2017 [7], sembra trovare il giusto equilibrio tra prestazioni e comprensione; inoltre, a differenza degli algoritmi trattati fino a ora in questa tesi, PPO permette di estendere lo spazio delle azioni dell'agente da discreto a continuo (tabella 2.1), consentendo quindi di aumentare il numero di problemi trattabili.

La sua principale caratteristica è quella di migliorare la stabilità dell'addestra-

mento dell'agente limitando le modifiche apportate alla policy a ogni epoca di addestramento evitando quindi aggiornamenti di policy troppo grandi durante l'addestramento. Questo è importante per due motivi, il primo è la **convergenza**, aggiornamenti di policy più piccoli durante l'addestramento tendono a convergere verso una soluzione ottimale, il secondo è la **stabilità**, aggiornamenti di policy troppo grandi possono portare a una politica che non funziona correttamente e rendere difficile o impossibile ottenere buoni risultati. Per misurare quanto la policy corrente è cambiata rispetto a quella precedente, viene utilizzato il calcolo del rapporto tra la policy corrente e quella precedente e questo determina il *Clipping* della funzione obiettivo. Il PPO utilizza una funzione obiettivo "clippata" per garantire che gli aggiornamenti di policy non siano troppo grandi, questo significa che il rapporto tra le policy viene limitato in un intervallo specifico. Se il rapporto è al di fuori di questo intervallo, viene "clippato" per garantire stabilità nell'addestramento.
Il PPO è basato su un'architettura actor-critic, che combina due componenti principali:

1. Attore (Actor): controlla il comportamento dell'agente e decide quali azioni intraprendere sulla base della policy corrente e il suo obiettivo è quello di massimizzare la ricompensa cumulativa nel tempo.

2. Critico (Critic): stima il valore degli stati e delle azioni e il suo obiettivo è quello di minimizzare l'errore tra la stima del valore e la ricompensa effettiva.

L'interazione tra attore e critico consente di bilanciare l'esplorazione dell'attore e l'apprendimento del critico, migliorando l'efficienza dell'addestramento [12].

Algorithm	**Action Space**	**State Space**
Q-Learning	Discrete	Discrete
DQN	Discrete	Discrete/Continuous
PPO	Discrete/Continuous	Discrete/Continuous

Tabella 2.1: Confronto tra lo spazio delle azioni e degli stati tra i vari algoritmi [19]

2.7 Introduzione a MatLab

Il nome **MATLAB**, acronimo di MATrix LABoratory, è un linguaggio di programmazione per il calcolo tecnico e scientifico. La sua caratteristica principale

è l'operare direttamente con matrici e non semplicemente con numeri, quindi i numeri e i vettori sono considerati come particolari matrici. Questo rende l'elaborazione di questo genere di input molto efficiente, rendendo noto tale ambiente negli ambiti universitari e lavorativi per le sue potenzialità di calcolo [17].
I *Toolbox* forniscono caratteristiche specifiche non presenti nel set di funzioni originali di MatLab e possono essere installati come moduli aggiuntivi. Uno tra questi è **Simcape** che consente di creare modelli di sistemi fisici all'interno dell'ambiente Simulink. Simulink è un ambiente di diagrammi a blocchi per la simulazione e la progettazione Model-Based costruito utilizzando i comandi di Matlab. Con Simscape possono essere costruiti modelli di componenti fisici basati su connessioni fisiche che si integrano direttamente con diagrammi a blocchi come motori elettrici, attuatori e sistemi di pulegge.

Capitolo 3

Implementazione del Workflow RL

In informatica, un **workflow** rappresenta una sequenza di attività o processi che devono essere eseguiti in un ordine specifico per raggiungere un obiettivo definito. Nel contesto di questo argomento, esploreremo come il linguaggio di programmazione MatLab semplifica le operazioni di Reinforcement Learning (RL), consentendo l'implementazione agevole di sistemi di controllo complessi. Inoltre, vedremo come la struttura di un agente RL rimane sorprendentemente simile per problemi estremamente diversi, anche in ambienti completamente differenti. Questo dimostra la flessibilità con cui un agente RL può adattarsi a vari contesti e risolvere differenti problemi, pur mantenendo una struttura coerente.
I passaggi principali che descriveranno il nostro workflow RL sono i seguenti:

1. Definizione dello spazio delle osservazioni e delle azioni

2. Definizione dell'ambiente e della reward

3. Definizione dell'algoritmo per l'agente e dei suoi iperparametri

4. Addestramento dell'agente

5. Simulazione e studio dei risultati

3.1 Implementazione di esempi di RL

Prima di esaminare un esempio di applicazione del workflow di un agente RL in un ambiente complesso, ci concentriamo su alcuni classici esempi della letteratura sul Reinforcement Learning, come il *CartPole* o il *Pong* in modalità singleplayer. In entrambi questi esempi, l'agente che implementeremo dovrà risolvere il problema cercando di massimizzare le rispettive ricompense. Nel primo esempio, utilizzeremo un ambiente predefinito fornito direttamente da MatLab, concentrandoci sulla costruzione dell'agente RL. Nel secondo esempio, invece, creeremo un ambiente personalizzato definendo sia le regole che le ricompense, e verificheremo il corretto funzionamento dell'agente RL.
Alla fine, noteremo come le strutture degli agenti RL si somiglino, nonostante gli obiettivi da raggiungere nei due problemi siano estremamente diversi.

3.1.1 Ambiente predefinito (CartPole)

Questo ambiente corrisponde alla versione del problema del CartPole descritto da Barto, Sutton e Anderson in "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problem "[3]. Un palo è fissato tramite un giunto non azionato a un carrello che si muove lungo un binario privo di attrito. Il pendolo è posizionato verticalmente sul carrello e l'obiettivo è bilanciare il palo applicando forze nella direzione sinistra e destra sul carrello.

Definizione dell'ambiente e della reward In questo esempio faremo uso di un ambiente predefinito fornito direttamente da MatLab. Analizzeremo due varianti dello stesso ambiente. La prima è:

```
env = rlPredefinedEnv("CartPole-Discrete");
```

dove l'insieme delle osservazioni e delle azioni è definito come discreto mentre la seconda è:

```
env = rlPredefinedEnv("CartPole-Continuous");
```

dove invece sia le azioni che le osservazioni sono continue. Viene fornita una ricompensa di +1 per ogni passo in cui il palo rimane in posizione verticale. Quando il palo cade, viene invece applicata una penalità di –5.

Definizione dello spazio delle osservazioni e delle azioni L'insieme delle azioni e delle osservazioni in questo esempio le ereditiamo direttamente dai parametri definiti all'interno dell'ambiente.

```
obsInfo = getObservationInfo(env);
actInfo = getActionInfo(env);
```

Definizione dell'algoritmo per l'agente e dei suoi iperparametri Per i due diversi ambienti, definiamo agenti specifici. Nel caso del *CartPole-Discrete*, abbiamo optato per la creazione di un agente che implementa l'algoritmo DQN. Nel caso del *CartPole-Continuous*, poiché l'insieme delle azioni è continuo, utilizzeremo l'algoritmo PPO.

Per l'implementazione del DQN-Agent avremo inizialmente bisogno di definire una DNN:

```
net = [featureInputLayer(obsInfo.Dimension(1))
    fullyConnectedLayer(20)
    reluLayer
    fullyConnectedLayer(length(actInfo.Elements))];
criticNet = dlnetwork(net);
critic = rlVectorQValueFunction(net, obsInfo, actInfo
   );
```

In questa sezione abbiamo definito una semplice DNN composta da 3 livelli e infine è stata assegnata come approssimatore per la funzione del Q-value. Successivamente è stato sufficiente definire gli iperparametri del nostro algoritmo e procedere alla creazione del nostro agente.

```
criticOpts = rlOptimizerOptions(...
    LearnRate=1e-3,...
    GradientThreshold=1);
agentOpts = rlDQNAgentOptions(...
CriticOptimizerOptions=criticOpts);
agent = rlDQNAgent(critic, agentOpts);
```

La scelta del valore assegnato a ciascun iperparametro deriva semplicemente da un processo iterativo di test per la selezione del migliore.

Per l'implementazione del PPO-Agent avremo invece bisogno di due differenti DNN, corrispondenti al critico e all'attore. La struttura della prima è simile a quella vista per l'agente precedente, mentre per la seconda richiede di creare un attore stocastico gaussiano, per cui è necessario utilizzare una DNN con due livelli di output: uno per la media e l'altro per la deviazione standard di ciascun componente dell'azione. L'attore per la selezione dell'azione campionerà da una distribuzione di probabilità gaussiana basata su questi output.

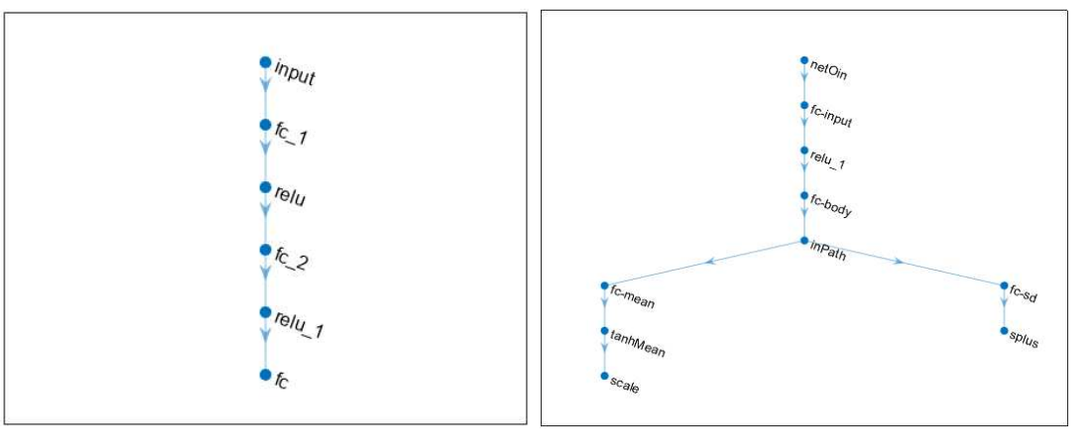

Figura 3.1: Struttura Critic-DNN e Actor-DNN

Infine, come per il precedente agente, sono stati definiti gli iperparametri ed è stato creato l'agente.

```
1 agent = rlPPOAgent(actor,critic);
2 agent.AgentOptions.ActorOptimizerOptions.LearnRate = 1e
     -3;
3 agent.AgentOptions.CriticOptimizerOptions.LearnRate = 1e
     -2;
```

Addestramento dell'agente Per addestrare l'agente abbiamo deciso di limitare a 1000 il numero massimo di episodi e di limitare a 500 gli steps per singolo episodio; questi valori sono stati inseriti all'interno di *trainOpts*. L'addestramento terminerà nel momento in cui la media delle reward negli ultimi 30 episodi è maggiore di 480.

```
1     trainingStats = train(agent,env,trainOpts);
```

Dai grafici 3.2 e 3.3, ottenuti al termine dell'addestramento dei rispettivi agenti DQN e PPO, possiamo notare che entrambi terminano l'addestramento prima del limite massimo di episodi, poiché raggiungono il loro obiettivo. In particolare, l'agente DQN riesce a raggiungere l'obiettivo di convergenza in un numero inferiore di episodi rispetto all'agente PPO. Questo risultato potrebbe essere attribuito al minor numero di variabili nelle osservazioni e al ridotto numero di azioni disponibili, che favoriscono l'approccio DQN.

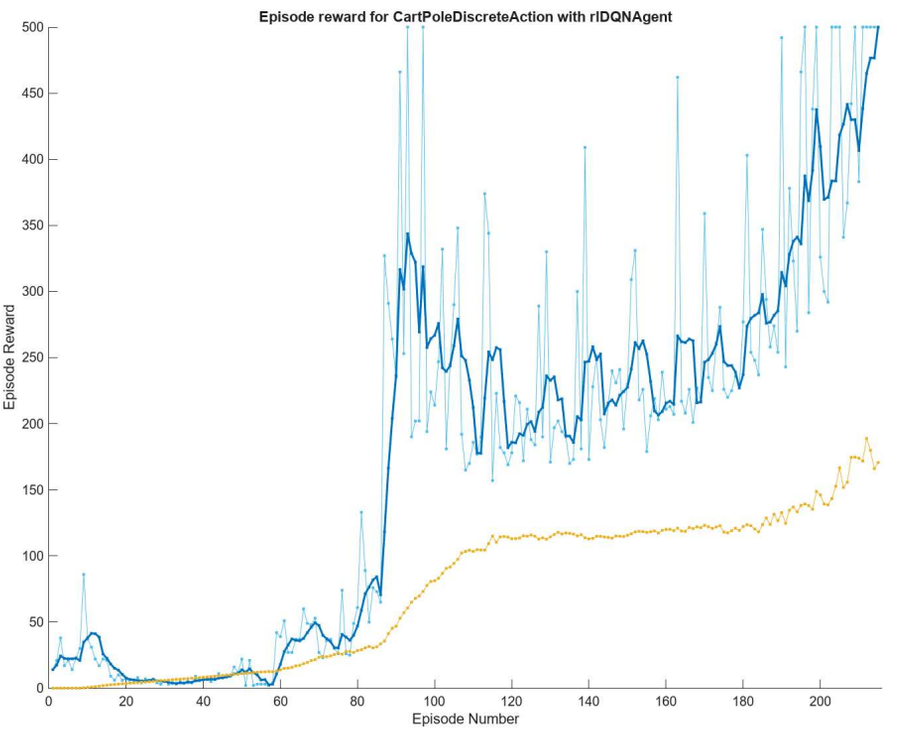

Figura 3.2: Grafico dell'addestramento dell'agente DQN

Simulazione e studio dei risultati Infine, abbiamo eseguito la simulazione per entrambi gli agenti, e in tutte e 20 le simulazioni, entrambi gli agenti sono riusciti a convergere e hanno ottenuto la massima precisione e accuratezza, come evidenziato nell'immagine 3.4

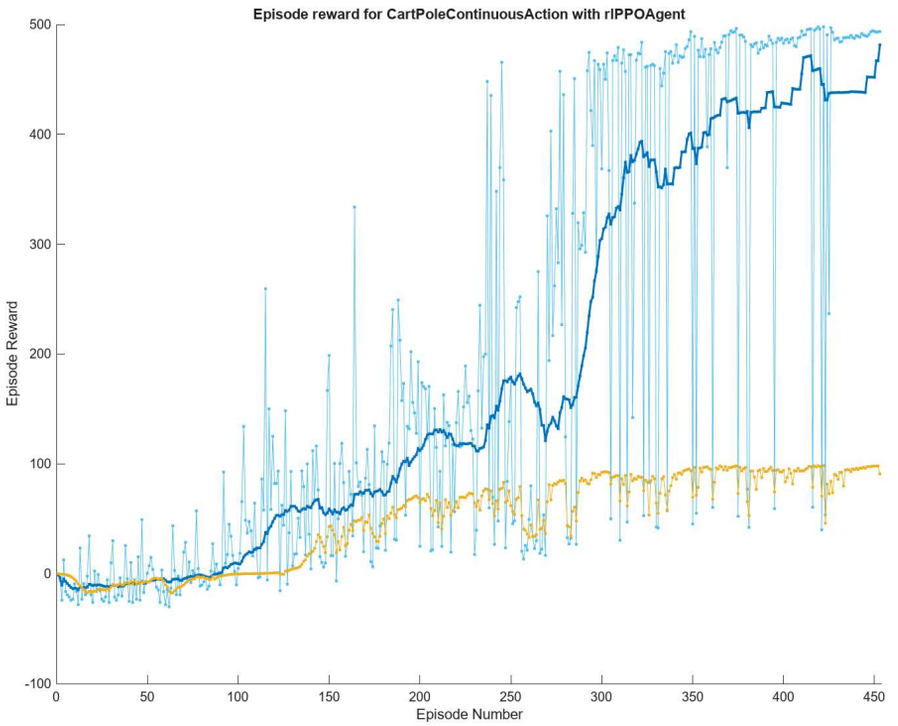

Figura 3.3: Grafico dell'addestramento dell'agente PPO

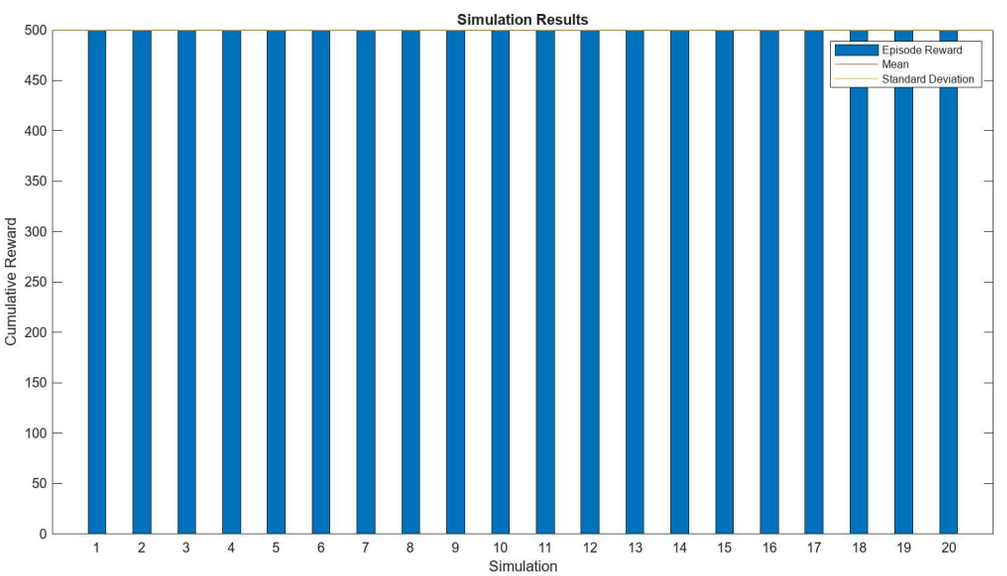

Figura 3.4: Risultato simulazione dell'agente PPO e DQN

3.1.2 Ambiente personalizzato (Pong)

Nel seguente esempio, esploreremo un classico gioco Atari: Pong in modalità singleplayer. In questo gioco, l'agente deve muovere una barra posta a sinistra del campo con l'obiettivo di intercettare la pallina e farla rimbalzare prima che finisca oltre la linea della barra. Concentreremo principalmente l'attenzione su come definire un ambiente personalizzato per l'agente.

Definizione dello spazio delle osservazioni e delle azioni L'insieme delle azioni e delle osservazioni è definito in due corrispondenti matrici:

```
obsInfo = rlNumericSpec([4 1]);
obsInfo.Name = "Pong States";
obsInfo.Description = 'yBar, xBall, yBall, theta';
actInfo = rlFiniteSetSpec([-0.7 0 0.7]);
actInfo.Name = "Pong Action";
```

Le variabili che compongono le osservazioni del gioco sono *yBar*, coordinata verticale della barra, *xBall*, coordinata orizzontale della palla, *yBall*, coordinata verticale (altezza) della palla, e *theta*, angolo di movimento della palla. Le variabili invece che compongono le azioni sono *0*, la barra rimane ferma, *0,7*, la barra si sposta verso l'alto di 0,7 unità, e *-0,7*, la barra si abbassa di 0,7 unità.

Definizione dell'ambiente e della reward Per definire un ambiente è necessario specificare due funzioni *resetFunction* e *stepFunction* che sono state implementate nei rispettivi file con estensione ".m".

```
type resetFunction.m
resetFnc = @resetFunction
type stepFunction.m
stepFnc = @stepFunction
env = rlFunctionEnv(obsInfo, actInfo, stepFnc,
   resetFnc);
```

La funzione resetFunction imposta l'ambiente su uno stato iniziale e calcola il valore iniziale dell'osservazione. Nel nostro esempio viene impostata la posizione centrale sia della pallina che della barra e infine viene calcolato un valore casuale per l'angolo di direzione della pallina.

```
function [InitialObservation, Info] = resetFunction()
    yBar = 15; xBall = 15; yBall = 15; r = rand;
    if r < 0.5
        theta = deg2rad(0) + ...
        (deg2rad(45)-deg2rad(0)).*rand;
    else
        theta = deg2rad(315) + ...
```

```
8               (deg2rad(360)-deg2rad(315)).*rand;
9           end
10          Info = [yBar;xBall;yBall;theta];
11          InitialObservation = Info;
12      end
```

La funzione stepFunction restituisce, invece, i valori dell'osservazione e della ricompensa successive, un valore logico che indica se l'episodio è terminato e una variabile con le informazioni dell'ambiente aggiornato. La funzione riceve come parametri due valori che indicano l'azione scelta dall'agente e lo stato corrente dell'ambiente. Nel nostro esempio inizialmente andiamo a definire alcune costanti che definiscono l'ambiente, come la grandezza del campo o della barra.

```
1       function [NextObs,Reward,IsDone,NextState] =
            stepFunction(Action,State)
2           velBall = 1; xBar = 1; lBar = 1;
3           hBar = 5;
4           lBall = 1;
5           lField = 30;
6           hField = 30;
7           Penalty = -20;
```

Successivamente aggiorniamo l'ambiente modificando le variabili e impostandole al nuovo valore rispetto all'azione intrapresa.

```
1           yBar = State(1)+Action;
2           xBall = State(2);
3           yBall = State(3);
4           theta = State(4);
5           xBall = xBall + velBall * cos(theta);
6           yBall = yBall + velBall * sin(theta);
```

Effettuiamo dei controlli per verificare eventuali rimbalzi che la pallina deve fare.

```
1           if yBar >= hField - hBar/2
2               yBar = hField - hBar/2;
3           end
4           if yBar <= hBar/2
5               yBar = hBar/2;
```

```
        end
        if xBall <= xBar + lBar && yBall <= yBar + hBar/2
            && yBall >= yBar - hBar/2
            theta = pi - theta;
        end
        if yBall <= 0 || yBall >= hField
            theta = -theta;
        end
        if xBall > lField
            theta = pi - theta;
        end
```

Infine definiamo le variabili contenenti i valori delle nuove osservazioni, il valore per determinare se l'episodio è terminato e infine calcoliamo il valore della ricompensa associata.

```
        NextState = [yBar; xBall; yBall; theta];
        NextObs = NextState;
        IsDone = xBall <= 0;
        if ~IsDone
            Reward = 1;
        else
            Reward = Penalty - abs(yBall - yBar);
        end
    end
```

Definizione dell'algoritmo per l'agente e dei suoi iperparametri In questo esempio abbiamo applicato l'algoritmo DQN utilizzando la stessa rete neurale utilizzata nell'esempio del capitolo 3.1.1; invece, durante la definizione degli iperparametri dell'agente sono stati aggiunti alcuni valori per implementare la Epsilon Greedy Strategy.

```
    agentOpts = rlDQNAgentOptions();
    agentOpts.EpsilonGreedyExploration.Epsilon = 1;
    agentOpts.EpsilonGreedyExploration.EpsilonMin =
       0.001;
```

```
4     agentOpts.EpsilonGreedyExploration.EpsilonDecay =
          0.001;
5     agentOpts.CriticOptimizerOptions.LearnRate = 1e-03;
6     agentOpts.DiscountFactor = 0.99;
7     agent = rlDQNAgent(critic, agentOpts);
```

Addestramento dell'agente Per l'addestramento dell'agente abbiamo limitato il numero massimo di episodi a 3000 e il numero massimo di steps per episodio a 700. Come possiamo notare dalla figura 3.5 l'agente dopo l'episodio 750 continua a oscillare senza mai raggiungere la convergenza.

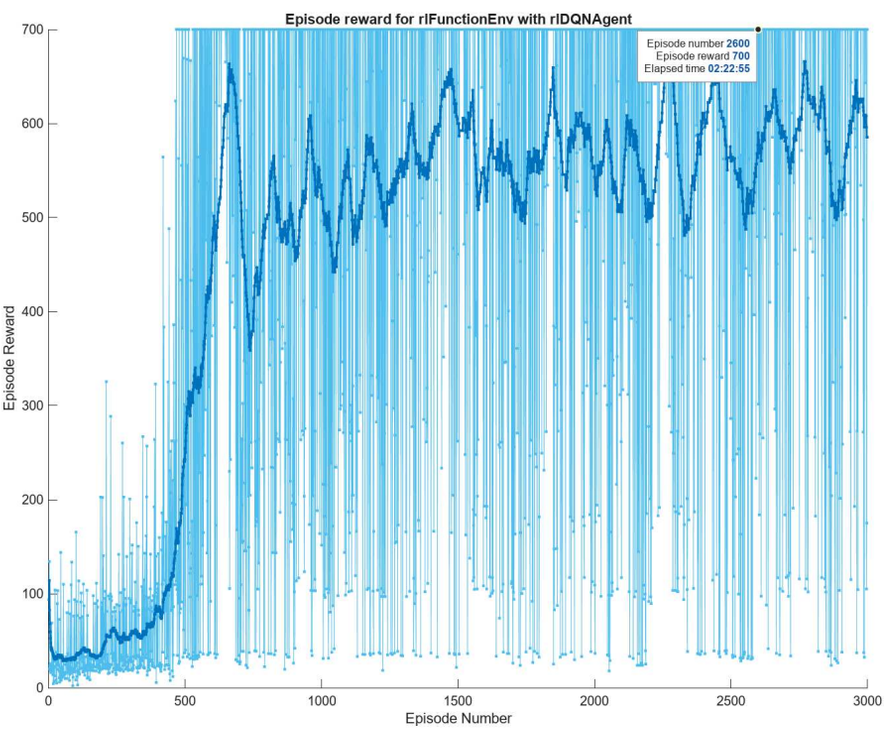

Figura 3.5: Grafico dell'addestramento dell'agente DQN

Simulazione e studio dei risultati Dai risultati delle simulazioni (figura 3.6) possiamo notare come l'agente non riesca ad avere una precisione massima, infatti è solo del 77% (77 partite vinte su 100). Probabilmente la causa è dovuta a un'assegnazione di una ricompensa non troppo precisa e a una rete neurale non configurata nel migliore dei modi.

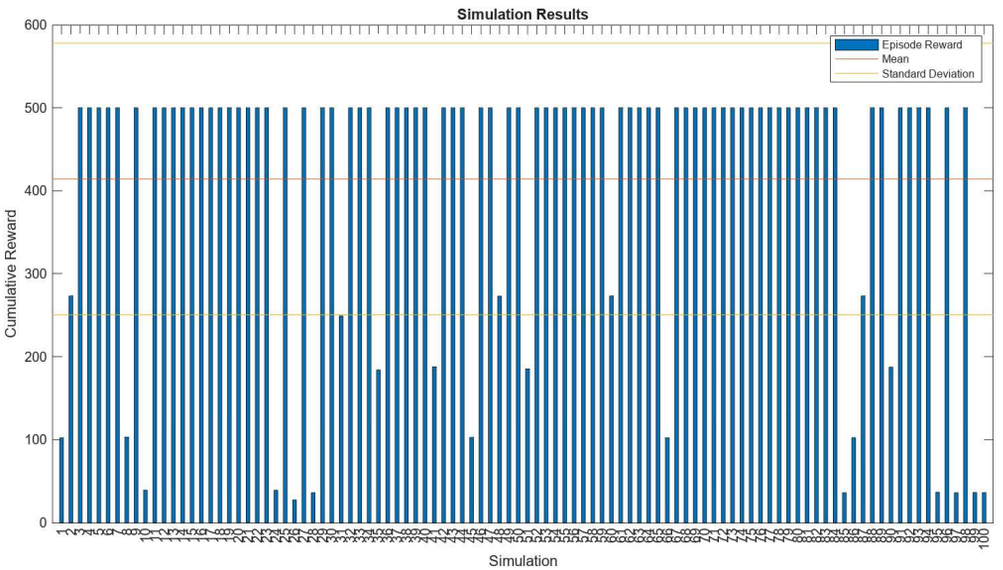

Figura 3.6: Risultato simulazione dell'agente PPO

3.1.3 Risultati degli esempi CartPole e Pong

Dai due esempi banali che abbiamo esaminato, possiamo notare come il processo di creazione dell'agente sia simile, nonostante l'utilizzo di ambienti diversi. Le operazioni e i passaggi richiesti sono gli stessi, ma la loro implementazione varia a seconda dell'ambiente specifico. Ora esploreremo un esempio più complesso che coinvolge una simulazione in Simscape e applicheremo lo stesso workflow che abbiamo appena analizzato.

3.2 Introduzione al Modello Simscape

Ora ci concentreremo su un problema che riguarda la creazione di un ottimizzatore di processo; questo strumento sfrutta la potenza del Reinforcement Learning (RL) per regolare automaticamente i parametri di processo di una macchina potenziale (come quella nella figura 3.7). Grazie all'apprendimento automatico, l'algoritmo dovrà esplorare in modo intelligente lo spazio dei parametri per massimizzare le prestazioni del processo, consentendo una produzione efficiente. Dato il problema abbiamo creato un modello in Simscape (come mostrato nella figura 3.8) che descriva la macchina rappresentata nella figura 3.7. Questo modello è costituito da due pulegge, che fungono da rappresentazione fedele delle pulegge reali. La loro dimensione può variare, ma l'interazione tra la cinghia e le pulegge è ideale. La puleggia A riceve una velocità di rotazione in radianti come input, mentre la

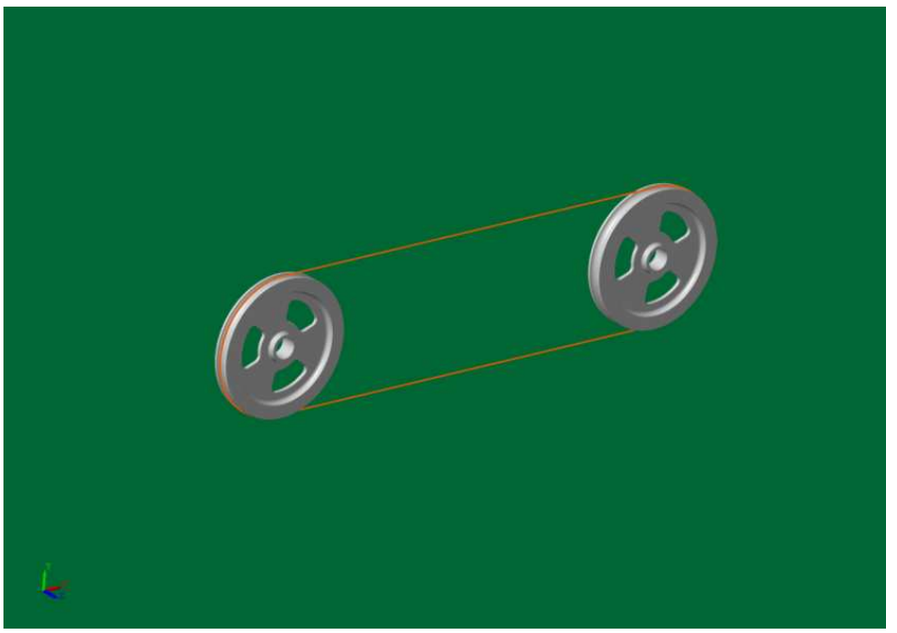

Figura 3.7: Modello Simscape 3D

puleggia B è libera di ruotare. Le due pulegge sono collegate da un cavo, e su di esso possiamo impostare diverse variabili come rigidezza o smorzamento.

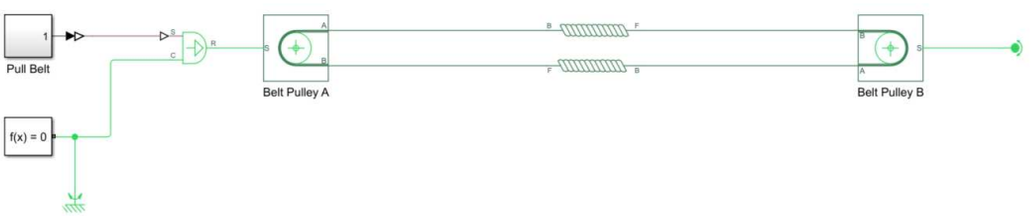

Figura 3.8: Modello Simscape

3.3 Implementazione dell'Agente RL per il modello Simscape

In questo capitolo, esamineremo gli agenti creati per affrontare il problema discusso nel capitolo 3.2, concentrandoci principalmente sul **workflow** e sulle differenze tra di essi. L'agente iniziale, versione 1.0, si propone di risolvere un problema semplificato, mentre nella versione 1.1 cercheremo di complicarlo ulteriormente introducendo nuove variabili. Infine, con l'ultimo agente, abbiamo affrontato il problema senza discretizzarlo, ma utilizzando direttamente un approccio continuo.

3.3.1 Versione 1.0 (DQN Agent)

Nel primo esempio di applicazione di Reinforcement Learning associato a questo modello, abbiamo scelto di simulare un sistema di pulegge utilizzando una velocità angolare costante per la puleggia A. Il nostro obiettivo era sviluppare un agente in grado di determinare la migliore rigidità (Stifness) del cavo, ignorando temporaneamente il valore di smorzamento, al fine di ottenere la forza minima nel cavo al termine della simulazione.

Definizione dello spazio delle osservazioni e delle azioni L'insieme delle osservazioni è costituito da due valori: la forza del cavo e la relativa rigidezza. Per quanto riguarda le azioni, abbiamo discretizzato l'insieme in modo che l'agente possa regolare la rigidità del cavo aumentandola o diminuendola di un'unità alla volta, in modo da permettergli di esplorare l'ambiente con l'obiettivo di trovare la rigidezza ottimale che corrisponda alla minima forza sul cavo.

```
obsInfo = rlNumericSpec([2 1]);
obsInfo.Name = "Model States";
obsInfo.Description = 'f_cable, stiffness_cable';
actInfo = rlFiniteSetSpec([-1 0 1]);
actInfo.Name = "Model Action";
```

Definizione dell'ambiente e della reward Come visto nel capitolo 3.1.2 abbiamo definito l'ambiente implementando due metodi: *resetFunction* e *stepFunction*. All'interno della *resetFunction*, abbiamo stabilito i parametri iniziali e inizializzato il valore di rigidità con un numero casuale.

```
function [InitialObservation, Info] = resetFunction()
     const_rot = 1;
     f_cable = 0;
     stiffness = randi([8, 30]);
     assignin('base', 'stiffness', stiffness);
    Info = [f_cable; stiffness];
    InitialObservation = Info;
end
```

Durante la *stepFunction*, invece, aggiorniamo il valore della rigidità in base all'azione intrapresa dall'agente e simuliamo nuovamente il modello per ottenere il nuovo valore della forza sul cavo.

```
function [NextObs,Reward,IsDone,NextState] = stepFunction
    (Action,State)
    f_cable = State(1);
    stiffness = State(2);
    stiffness = stiffness + Action;
    if stiffness > 100
        stiffness = 100;
    end
    if stiffness < 1
        stiffness = 1;
    end
```

Per simulare il nostro modello è sufficiente richiamare la funzione *sim* che ci restituirà al termine un vettore Force contenente tutte le forze misurate dal sensore posto sul cavo. Tramite la funzione *assignin* invece è possibile impostare il valore dei parametri nel modello.

```
    assignin('base', 'stiffness', stiffness);
    model = "agent_1.0/model_test_2pulley.slx";
    load_system(model);
    sim(model);
    f_cable = max(Force)
    NextState = [f_cable; stiffness];
    NextObs = NextState;
    Reward = -abs(f_cable);
    IsDone = false;
end
```

Il valore della ricompensa associata all'azione viene quindi calcolato tramite l'uso della funzione continua del valore assoluto che determina lo spostamento rispetto al nostro valore atteso.

Definizione dell'algoritmo per l'agente e dei suoi iperparametri Per l'implementazione del nostro primo agente, abbiamo scelto l'algoritmo DQN (Deep

Q-Network) poiché l'insieme delle azioni è discreto, mentre l'insieme delle osservazioni è continuo. La rete neurale profonda (DNN) associata è stata la stessa di quella implementata nel capitolo 3.1.1, con l'unica differenza che abbiamo aumentato il numero di nodi nel livello centrale della rete da 20 a 32.
Di seguito, riportiamo la definizione dell'agente e dei suoi iperparametri associati:

```
agentOpts = rlDQNAgentOptions();
agentOpts.EpsilonGreedyExploration.Epsilon = 1;
agentOpts.EpsilonGreedyExploration.EpsilonMin = 0.001;
agentOpts.EpsilonGreedyExploration.EpsilonDecay = 0.01;
agentOpts.CriticOptimizerOptions.LearnRate = 0.01;
agentOpts.DiscountFactor = 0.1;
agent = rlDQNAgent(critic, agentOpts);
```

Addestramento dell'agente Durante l'addestramento dell'agente, abbiamo eseguito solo pochi episodi con un numero limitato di passaggi. Questo è stato limitato poiché l'esecuzione della simulazione richiedeva un tempo considerevole, e le macchine su cui abbiamo effettuato l'addestramento avevano delle limitazioni in termini di potenza computazionale.

Simulazione e studio dei risultati Analizzando i risultati dell'addestramento, grafico 3.9, possiamo notare come l'agente ha ottenuto una ricompensa media finale di -7. Questi dati suggeriscono un buon risultato, anche se ammettono ancora un margine di errore, ma ciò non rappresenta un ostacolo in questa fase per il proseguimento. La simulazione non è sempre soddisfacente a causa di alcuni problemi come l'accuratezza e la precisione del risultato e ipotizziamo che le cause possano essere le seguenti: la scarsa presenza di casualità rende difficile per la rete neurale calcolare l'azione ottimale, una configurazione errata della rete neurale, iperparametri dell'agente non adatti al problema e la scelta dell'agente DQN limita l'impostazione di azioni discrete e questo porta a una bassa accuratezza del modello. Inoltre, il basso numero di steps per episodio e il ridotto numero di episodi di addestramento limitano sicuramente l'apprendimento dell'agente.

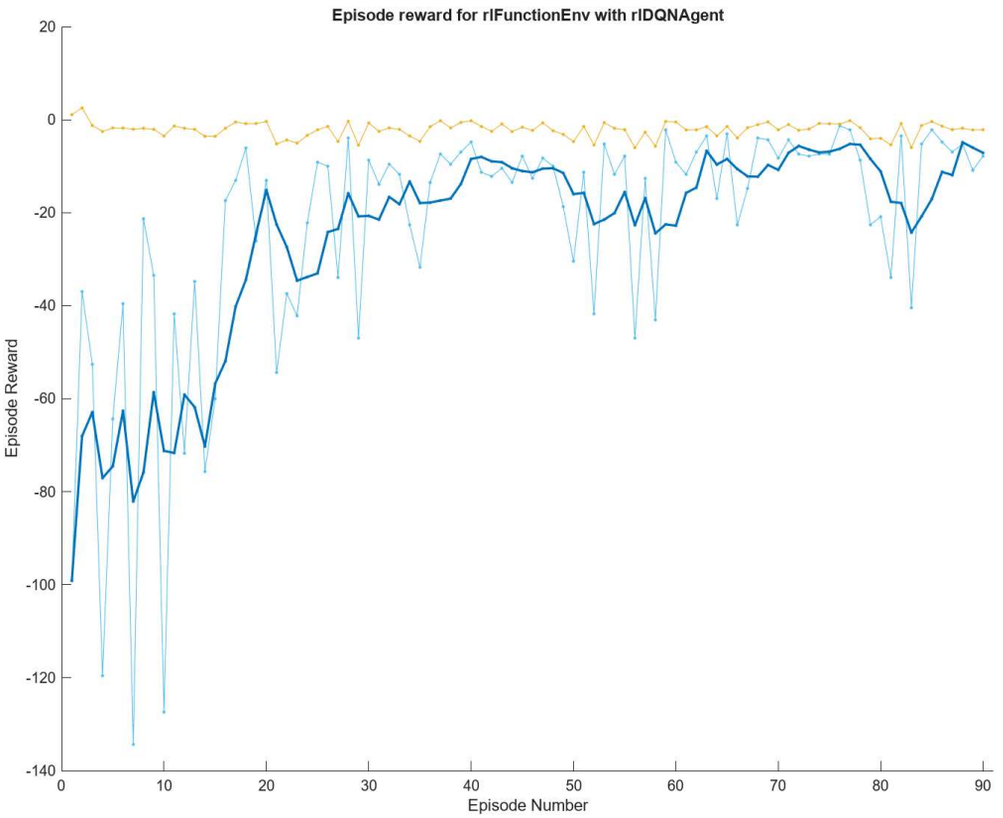

Figura 3.9: Addestramento agente 1.0

3.3.2 Versione 1.1 (DQN Agent)

Rispetto alla versione 1.0, abbiamo introdotto una velocità di rotazione casuale associata alla puleggia A, con valori compresi tra 0,5 e 2. Inoltre, abbiamo migliorato la discretizzazione delle azioni che l'agente può compiere, ampliando le possibilità di incremento e diminuzione dei valori di rigidità. Questi cambiamenti mirano a ottimizzare il comportamento dell'agente e a rendere la simulazione più accurata.

Definizione dello spazio delle osservazioni e delle azioni Abbiamo quindi introdotto una nuova variabile all'insieme delle osservazioni introducendo la velocità iniziale di rotazione della puleggia e inoltre abbiamo aumentato la discretizzazione delle azioni.

```
obsInfo = rlNumericSpec([3 1]);
obsInfo.Name = "Model States";
obsInfo.Description = 'f_cable, stiffness_cable,
   const_rot';
```

```
actInfo = rlFiniteSetSpec([-10 -5 -1 -0.5 -0.1 0 0.1 0.5
    1 5 10]);
actInfo.Name = "Model Action ";
```

Definizione dell'ambiente e della reward Nella definizione dell'ambiente e nella gestione delle ricompense, abbiamo mantenuto la stessa struttura vista in precedenza. Tuttavia, durante la fase di reset, abbiamo introdotto il nuovo parametro di rotazione, inizializzandolo con un valore casuale.

Definizione dell'algoritmo per l'agente e dei suoi iperparametri L'algoritmo utilizzato per l'agente è rimasto invariato rispetto alla versione precedente.

Addestramento dell'agente Anche la fase di addestramento è rimasta invariata. Abbiamo eseguito lo stesso numero di steps per episodio.

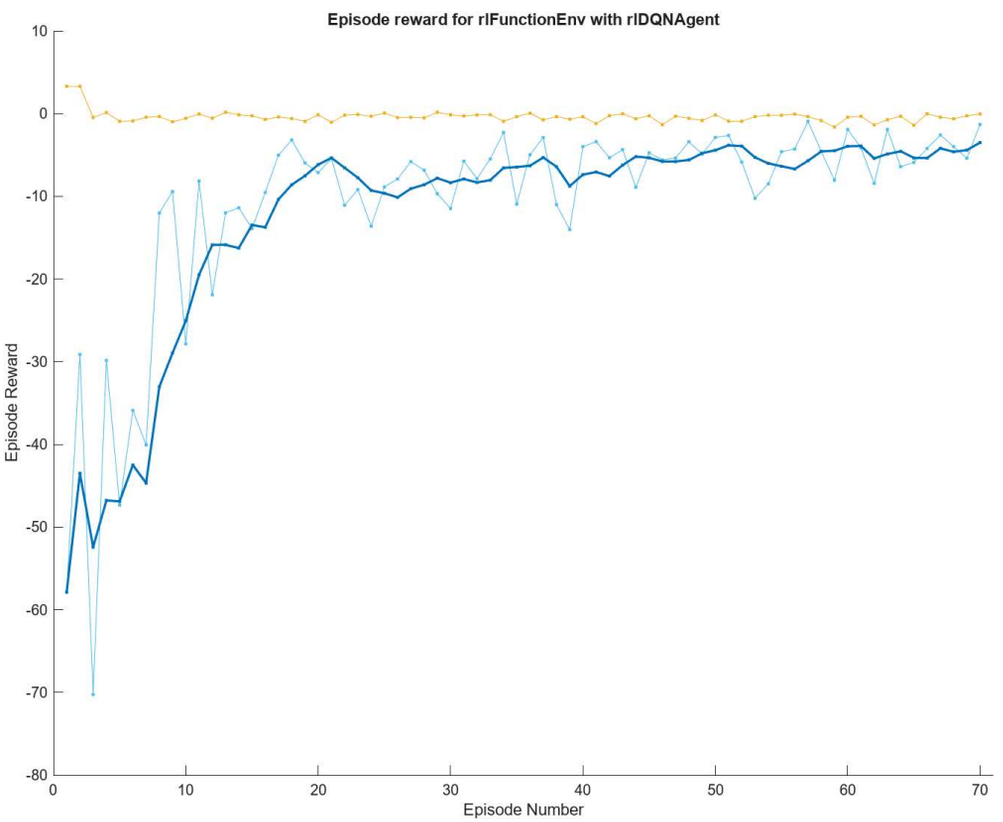

Figura 3.10: Addestramento agente 1.1

Simulazione e studio dei risultati Rispetto al modello 1.0, il nostro attuale approccio ci consente di ottenere una stima più precisa della rigidità in meno iterazioni, come evidenzia il grafico 3.10 che differisce rispetto al grafico 3.9. per

una maggiore stabilità. Questo risultato potrebbe derivare dalla maggiore discretizzazione delle azioni che l'agente può intraprendere. Inoltre, l'introduzione di una nuova variabile, come la velocità di rotazione della puleggia, non sembra aver avuto effetti negativi sull'apprendimento dell'agente.

3.3.3 Versione 2.0 (PPO Agent)

Affronteremo la stessa problematica descritta nel capitolo precedente, ma questa volta in un ambiente continuo, utilizzando l'algoritmo PPO per risolverla.

Definizione dello spazio delle osservazioni e delle azioni L'insieme delle azioni che l'agente può intraprendere è inizializzato come un insieme continuo di numeri compresi tra 10 e 30 che rappresentano i possibili valori della rigidità del cavo.

```
obsInfo = rlNumericSpec([3 1]);
obsInfo.Name = "Model States";
obsInfo.Description = 'f_cable, stiffness_cable,
    const_rot';
actInfo = rlNumericSpec([1 1],'LowerLimit',10,'UpperLimit
    ',30);
actInfo.Name = "Model Action";
```

Definizione dell'ambiente e della reward La definizione dell'ambiente è rimasta invariata rispetto la versione precedente.

Definizione dell'algoritmo per l'agente e dei suoi iperparametri L'agente PPO che abbiamo sviluppato utilizza lo stesso schema di rete neurale profonda (DNN) presentato nel capitolo 3.1.1 durante la creazione dell'agente PPO. Inoltre, abbiamo definito gli iperparametri nel seguente modo:

```
agent = rlPPOAgent(actor,critic);
agent.AgentOptions.ActorOptimizerOptions.LearnRate = 1e
    -3;
agent.AgentOptions.CriticOptimizerOptions.LearnRate = 1e
    -2;
agent.AgentOptions.MiniBatchSize = 32;
```

```
agent.AgentOptions.ExperienceHorizon = 32;
agent.AgentOptions.DiscountFactor = 0.2;
```

Addestramento dell'agente La fase di addestramento ha subito solo un incremento significativo del numero di episodi, mantenendo però lo stesso rapporto di numero di steps per episodio.

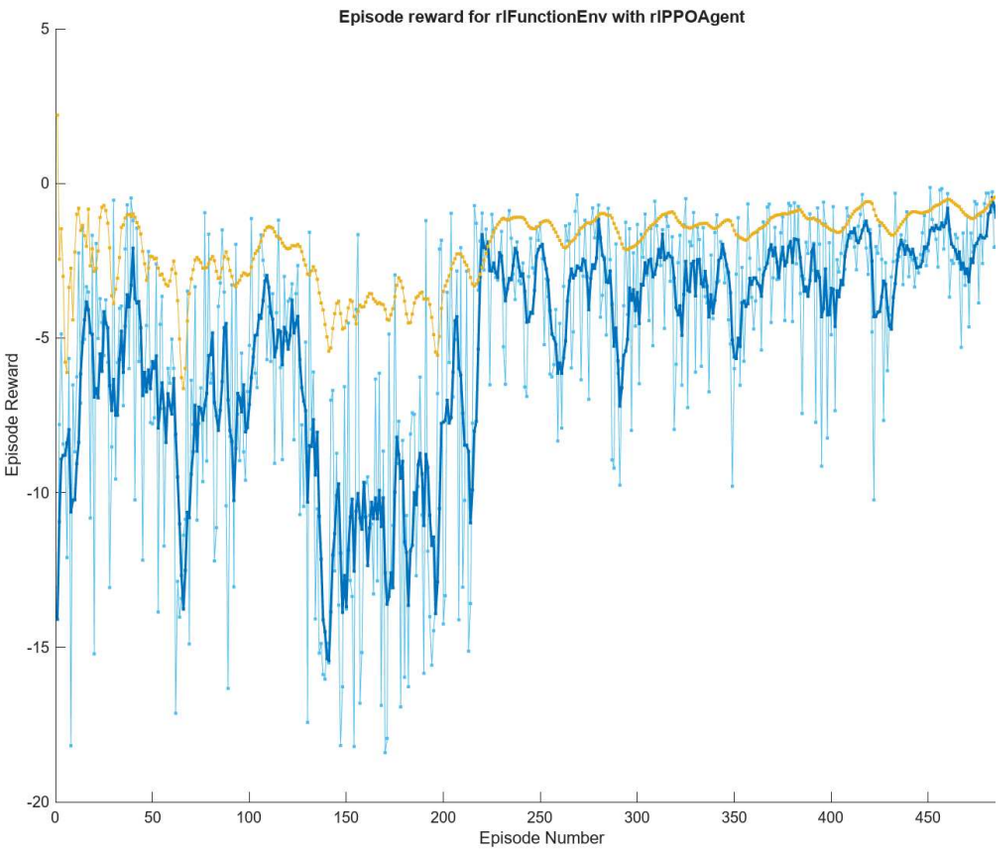

Figura 3.11: Addestramento agente 2.0

Simulazione e studio dei risultati L'agente risultante, sebbene non raggiunga un risultato ottimo, dimostra comunque prestazioni sufficienti. Nel confronto tra l'agente PPO (2.0) e l'agente DQN (1.1), notiamo però una diminuzione dell'accuratezza (figura 3.12). Questa discrepanza potrebbe derivare principalmente dalla mancanza di un'ottimizzazione efficace degli iperparametri. Pertanto, il prossimo passo nel nostro processo di sviluppo dovrà prevedere l'implementazione di uno strumento specifico per ottimizzare tali parametri, al fine di migliorare ulteriormente le prestazioni dell'agente.

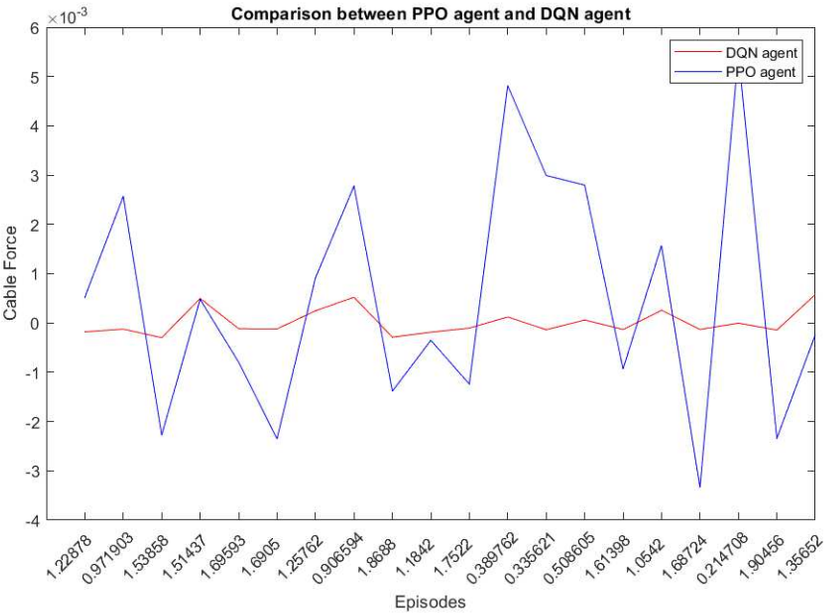

Figura 3.12: Comparazione della accuratezza tra gli agenti 1.1 e 2.0

Capitolo 4

Conclusioni

La tesi esplora l'applicazione del Reinforcement Learning (RL) e dei Digital Twin nell'ambito industriale, evidenziando come queste tecnologie possano supportare decisioni strategiche e ottimizzare i processi. Attraverso l'implementazione di un workflow basato sul RL in MatLab, la tesi mostra una possibile soluzione che mira a migliorare l'efficienza dei sistemi industriali.

La tesi esplora, inoltre, come un workflow di RL possa essere applicato in modo flessibile a diversi problemi dimostrando la consistenza di esso, nonostante la diversità dei problemi affrontati, sottolineando l'adattabilità del RL e la sua applicabilità in vari contesti.

I risultati ottenuti dalla tesi includono lo sviluppo di un workflow per un agente di Reinforcement Learning che fornisce supporto decisionale a un modello di Digital Twin implementato in MatLab. Un ulteriore sviluppo è la realizzazione di esempi di RL con ambienti predefiniti e personalizzati, che hanno dimostrato la flessibilità e l'adattabilità dell'agente RL in vari contesti. Infine, l'agente di Reinforcement Learning può migliorare le prestazioni del sistema attraverso l'apprendimento dalle proprie azioni e l'analisi dei dati direttamente da un possibile Digital Twin, che si è rivelato utile come modello virtuale per simulazioni e ottimizzazioni che possono migliorare l'efficienza del possibile sistema reale corrispondente.

La tesi sottolinea l'importanza del RL nell'industria 4.0, dove può automatizzare processi, migliorare la produttività e supportare decisioni strategiche. La sua flessibilità consente di adattarsi a diversi contesti industriali, mantenendo un approccio costante. L'uso di ambienti simulati accelera l'apprendimento e permette di testare scenari complessi, contribuendo all'ottimizzazione delle risorse e dei processi.

Durante lo sviluppo della tesi sono state incontrate alcune difficoltà. In primo luogo la bassa potenza computazionale a disposizione ha limitato spesso l'addestramento dell'agente. Non aver fatto uso di strumenti per l'ottimizzazione degli iperparametri degli agenti anche esso ha limitato l'addestramento di alcuni agenti. Infine l'ambiente Simscape utilizzato per il modello simulato delle pulegge si è rivelato limitato e le basse conoscenze di meccanica industriale ci hanno limitato sulla complessità e sui possibili vincoli assegnati al modello.

In sintesi, la tesi illustra come il RL possa essere un potente strumento per l'innovazione nell'industria, con un workflow che rimane consistente attraverso diversi problemi, facilitando l'adozione di questa tecnologia in una vasta gamma di applicazioni.

In conclusione, questa tesi conferma il potenziale del Reinforcement Learning e dei Digital Twin come strumenti utili per l'Industria 4.0, capaci di trasformare i dati in azioni strategiche e di aiutare l'evoluzione verso sistemi produttivi più intelligenti e autonomi.

Bibliografia

[1] A definition of ai: Main capabilities and scientific disciplines. 18 December 2018.

[2] Pyeman J Othman AK Sorooshian S Abideen AZ, Sundram VPK. Digital twin integrated reinforced learning in supply chain and logistics. *Logistics*, 2021. URL https://doi.org/10.3390/logistics5040084.

[3] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, pages 834–846, 1983. URL https://ieeexplore.ieee.org/document/6313077.

[4] Michele Buzzoni. Reinforcement learning in problemi di controllo del bilanciamento, 2017.

[5] Jens Kober Cosimo Della Santina Zlatan Ajanović Eveline Drijver, Rodrigo Pérez-Dattari. Robotic packaging optimization with reinforcement learning. *arXiv*, 2023.

[6] Stefan Wegenkittl Georg Sch¨afer, Reuf Kozlica and Stefan Huber. *An Architecture for Deploying Reinforcement Learning in Industrial Environments*. Springer Nature Switzerland, 2022.

[7] Prafulla Dhariwal Alec Radford Oleg Klimov John Schulman, Filip Wolski. Proximal policy optimization algorithms. 2017.

[8] Cieri Lorenzo. L'importanza del digital twin nell'industria 4.0, 2020.

[9] Batta Mahesh. Machine learning algorithms - a review. *International Journal of Science and Research (IJSR)*, 2020.

[10] oracle. Cos'è il machine learning? URL https://www.oracle.com/it/artificial-intelligence/machine-learning/what-is-machine-learning/.

[11] Marcel Panzer and Benedict Bender. Deep reinforcement learning in production systems: a systematic literature review. *International Journal of Production Research*, 2022.

[12] Jan Peters and Stefan Schaal. Natural actor-critic. *Neurocomputing*, 71:1180–1190, 2008.

[13] Martin L. Puterman. Chapter 8 markov decision processes. In *Stochastic Models*, volume 2 of *Handbooks in Operations Research and Management Science*, pages 331–434. Elsevier, 1990.

[14] Zhiyong Liu Chao Su Bo Wang Qingfei Min, Yangguang Lu. Machine learning based digital twin framework for production optimization in petrochemical industry. *International Journal of Information Management*, 2019.

[15] M. Mazhar Rathore, Syed Attique Shah, Dhirendra Shukla, Elmahdi Bentafat, and Spiridon Bakiras. The role of ai, machine learning, and big data in digital twinning: A systematic literature review, challenges, and opportunities. *IEEE Access*, 2021.

[16] Biao Huang Rui Nian, Jinfeng Liu. A review on reinforcement learning: Introduction and applications in industrial process control. *Computers Chemical Engineering*, 2020.

[17] Enzo Tonti. Introduzione a matlab. 2003.

[18] Marco A Wiering and Martijn Van Otterlo. Reinforcement learning. *Adaptation, learning, and optimization*, 12(3):729, 2012.

[19] Jie Zhu, Fengge Wu, and Junsuo Zhao. An overview of the action space for deep reinforcement learning. *Proceedings of the 2021 4th International Conference on Algorithms, Computing and Artificial Intelligence*, 2021.